**Design Exercise Report**

**on**

# Self-Tuning Extremum Control by Kittisak Tiyapan

**under the supervision of**

**Dr M B Zarrop**

This report is a part of the MSc Course in Control and Information Technology, 1994/95.

Control System Centre,

University of Manchester Institute of Science and Technology,

Manchester M60 1QD.

## Abstract

Self-tuning extremum control is useful when dealing with systems with performance index measured in a noisy environment. This design-exercise studies the approach using the model-based hill-climbing technique . The recursive least square (RLS) algorithm is used. The model used is a quadratic one. The idea is then used in object-location .

# Contents

## List of Figures

# 1 Introduction

## 1.1 Important terms

The term *extremum control* [1] is a synonym for optimization . It means making a process operate at the optimal point at all times. The optimal point is normally obtained by extremizing a performance index.

*Hill-climbing* is a very useful technique. We can imagine a performance function as being a hill. Figure 1 shows an example of a performance hill. When we are at a specific point

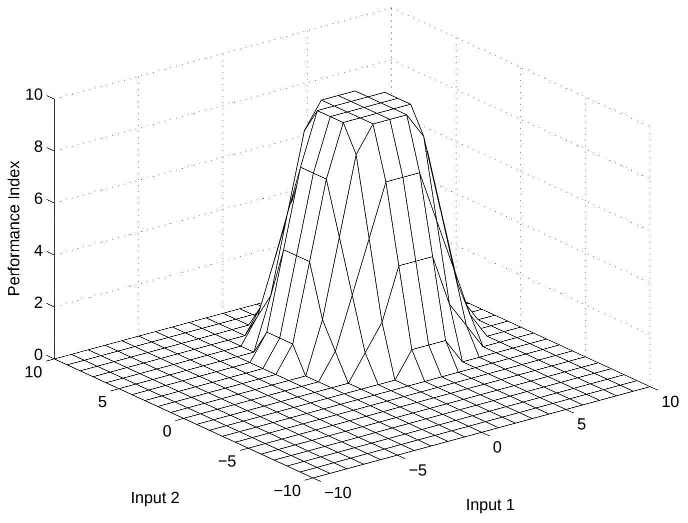


Figure 1: *Example of a relationship between a performance index and two inputs*

(control value) we can calculate the performance index at that point. The performance index we obtain represents the height of the hill there. If we compare this performance index with those of the points in our close neighbourhood we will know which direction to move to in order to increase our performance index. The process is carried out recursively until we reach the top of the hill where no neighbouring point leads to an improvement in performance.

*Self-tuning control* [1] is closely related to adaptive control. Here the input and output of the system are measured and used to adjust model parameters.

When there exists some reference model describing the performance, the output from the reference model, together with measured input/output data can be used to monitor system parameters. This is called a *model-based* algorithm [1] .

## 1.2 Objectives

- Investigate various hill-climbing algorithms.
- Apply algorithms to multi-input case .
- Apply algorithms to object location problem .

## 1.3 Overview of tasks

Performance optimization can be done using hill-climbing technique . The environment is usually noisy where the noise is either sensor noise or represents the roughness of the hill.

In the model-based approach used here a quadratic model is chosen. It is possible to reduce some parameters in the model [1].

With the recursive least squares algorithm it is possible to extend the approach to multi-input case [2] and hence to an object-location problem .

The system noise is assumed to be a gaussian random variable with zero mean and adjustable variance. To prevent the estimator from *sleeping* a dither signal is added which, in this case, is a zero-mean uncorrelated process, uniformly distributed on a chosen interval.

# 2 Simulations

## 2.1 Quadratic model

### 2.1.1 Introduction

In the neighbourhood of extremum a quadratic model is a good representation of the hill top. If the hill is smooth at the summit, this model represents the first few terms of a Taylor's series for the performance function. The model chosen is

$$M1: \quad y = au^2 + bu + c \tag{1}$$

where $y$ is the performance index to be maximized and $u$ is the adjustable factor (control input). If $a < 0$, then (1) represents a hill (not a valley) with a summit at

$$u = -\frac{b}{2a} \tag{2}$$

The true relationship between $u$ and $y$ is only approximated by (1). The true hill is usually mapped out empirically but, for simulation purposes, we may assume that the *data generator* is given by

$$y = g(u) + noise \tag{3}$$

where the noise represents either sensor noise or reflects the *roughness* of the hill. In simulations, (3) is implemented as a subroutine and the noise is generated as a gaussian random variable with zero mean and adjustable variance.

An iterative algorithm for finding the hill summit is as follows. (Here $t$ denotes the iteration count and the input $u(t-1)$ gives rise to the next output y(t).)

### 2.1.2 Quadratic model algorithm (Model M1)

**Initial Conditions** Set $t = 0, \hat{a}(0), \hat{b}(0), \hat{c}(0), \mathbf{P}_{(3\times3)}(0)$

**Step 1** Calculate

$$u(t) = -\frac{\hat{b}(t)}{2\hat{a}(t)} + v(t) \tag{4}$$

**Step 2**

$$\begin{aligned} t &= t+1 & (5)\\ y(t) &= g(u(t-1)) + e(t) & (6) \end{aligned}$$

**Step 3**

$$\begin{aligned} \mathbf{P}(t) &= \frac{1}{\lambda}\mathbf{P}(t-1)\left\{\mathbf{I}_3 - \frac{\mathbf{x}(t)\mathbf{x}^T(t)\mathbf{P}(t-1)}{\lambda + \mathbf{x}^T(t)\mathbf{P}(t-1)\mathbf{x}(t)}\right\} & (7)\\ \hat{\theta}(t) &= \hat{\theta}(t-1) + \mathbf{P}(t)\mathbf{x}(t)\left\{y(t) - \mathbf{x}^T(t)\hat{\theta}(t-1)\right\} & (8) \end{aligned}$$

**Step 4** Goto Step 1

**NB**

$$\begin{aligned} \hat{\theta}(t) &= \left[\hat{a}(t), \hat{b}(t), \hat{c}(t)\right]^T \\ \mathbf{x}(t) &= \left[u^2(t-1), u(t-1), 1\right]^T \end{aligned}$$

**Remark** In (4) the current parameter estimates are inserted in (2) to give an estimate of the optimal input. The additional *dither* signal $v(t)$ is used to excite the system, so that the estimator does not *go to sleep.* The dither signal is chosen as a zero mean uncorrelated process, uniformly distributed on a chosen interval.

### 2.1.3 Quadratic model simulation

Try the algorithm using various conditions and with different types of hill. Estimating hills using a quadratic model.

Basis : $\lambda = 0.98$, $\sigma_v^2 = 1.0$, $\sigma_e^2 = 0.5$

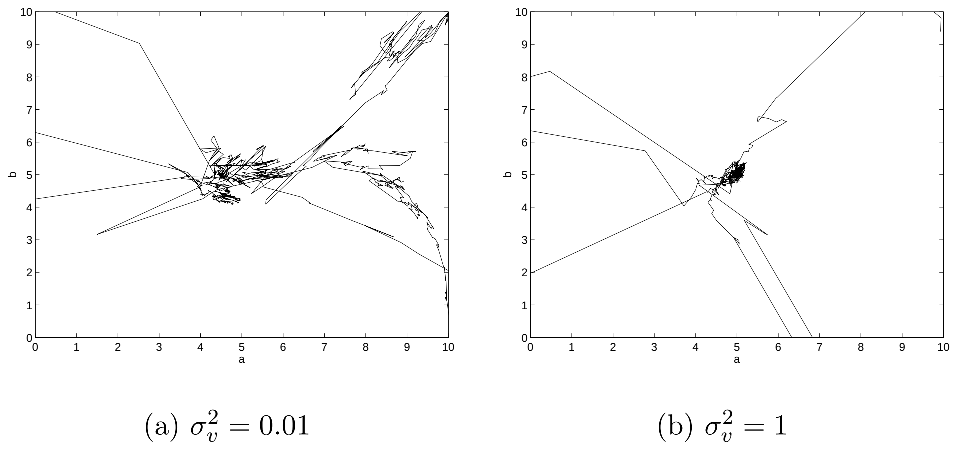
($\lambda$ = forgetting factor, $\sigma_v^2$ = dither signal's variance, $\sigma_e^2$ = noise's variance )

### 2.1.4 Results

The result does not depend on initial position of estimated hill top.

**When** $\lambda = 0.10, 0.50, 0.70, 0.90, 0.95, 0.98, 0.99, 1.0, 3.0$ **were used.** $\lambda$ between 0.95 and 1.0 gives good result. Smaller $\lambda$ produces much uncertainty. When $\lambda > 1$ is used the result does not converge to the true value.

**When** $\sigma_v^2 = 0.1, 0.3, 0.5, 0.7, 0.9, 1, 3, 5, 7, 9$ **were used.** Time of convergence is generally approximately 50 time steps. When $\sigma_v^2$ is small, the parameters' values estimated will scatter around the true value. This can be easier illustrated using a plot between two parameters. Figure 2 shows a comparison between two different values of $\sigma_v^2$'s.



(a) $\sigma_v^2 = 0.01$ (b) $\sigma_v^2 = 1$

Figure 2: *Plot of* $\hat{b}(t)$ *vs.* $\hat{a}(t)$*: (a)* $\sigma_v^2 = 0.01$ *, (b)* $\sigma_v^2 = 1$

**When various hill types are used.** (Hills used : circular-top, sinusoidal, third order polynomial, fourth order polynomial, sinc function, symmetric piece-wise functions, asymmetric piece-wise function, pulse shaped function.) Convergence to the local extremum that is closest to starting point. For badly conditioned hill, for example a pulse shaped function, convergence does not always occur within 300 time steps. Bias exists with all non-symmetric hills.

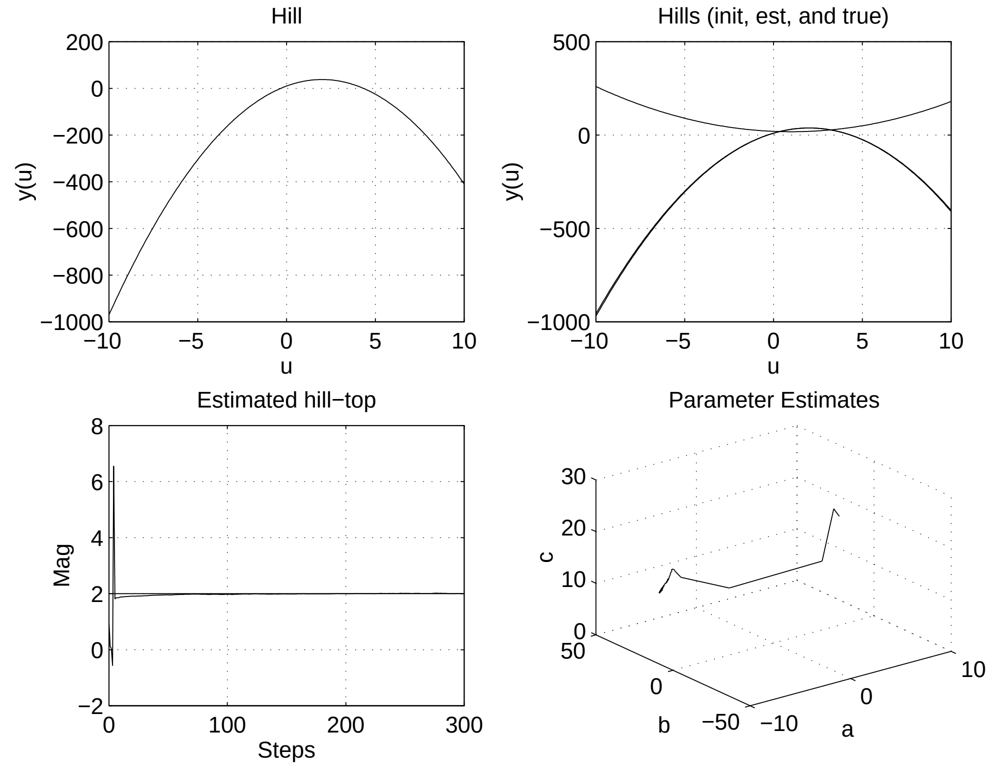


Figure 3: *Model M1 with quadratic system and model*

Figure 3: When both system and model are quadratic estimation is most perfect. Here we have the true hill (top left). We start off with a quadratic estimating valley and end up with exactly the same hill (top right). Control input is plotted against number of steps (bottom left). Estimated parameters $\hat{a}$, $\hat{b}$, and $\hat{c}$ are plotted to show the convergence process during estimation (bottom right).

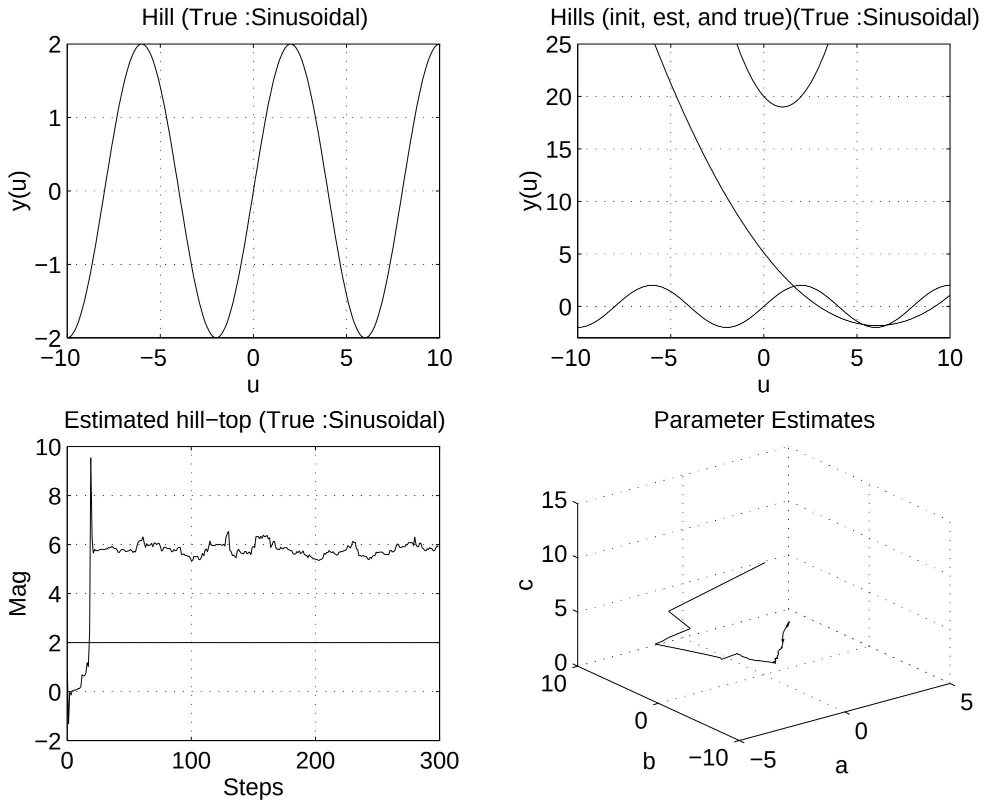


Figure 4: *Model M1 with sinusoidal-shaped hill*

Figure 4: This is the case where the true hill is sinusoidal. Here we have the true hill (top left). We start off with a quadratic estimating valley and end up with another valley (top right). Control input is plotted against number of steps (bottom left). Notice that in this case the hill top estimated turns out to be a local minimum. Estimated parameters $\hat{a}$, $\hat{b}$, and $\hat{c}$ are plotted to show the convergence process during estimation (bottom right).

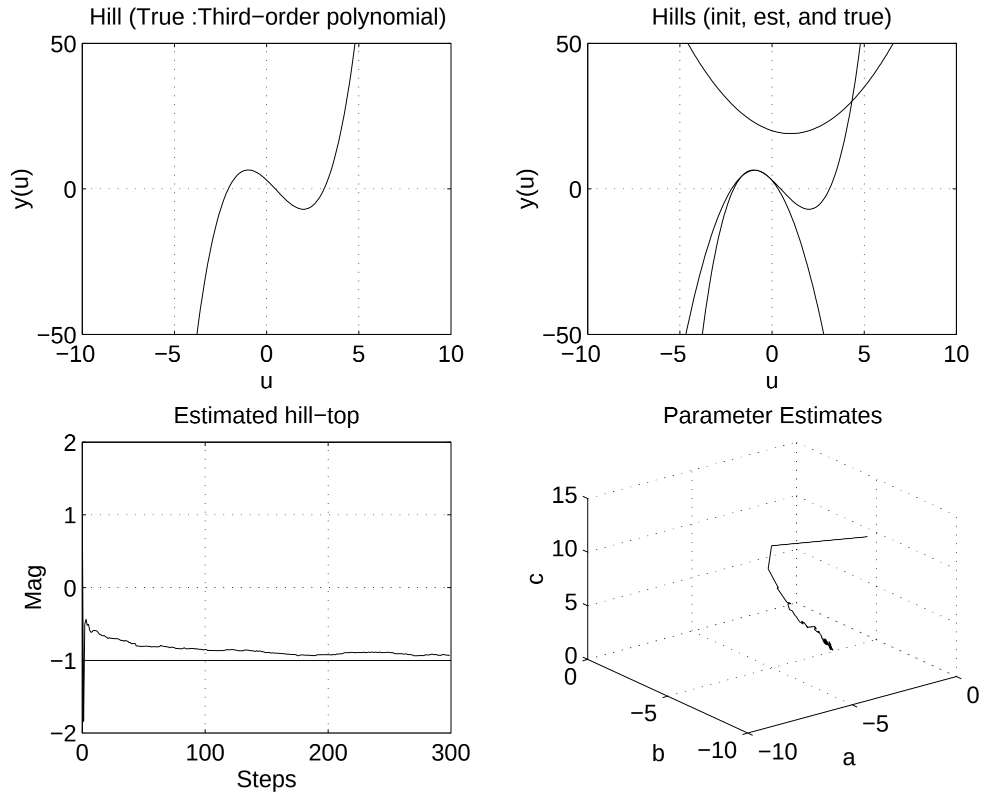


Figure 5: *Model M1 with a third order polynomial hill*

Figure 5: This is the case where the true hill is a third order polynomial. Here we can see a plot of the true hill (top left). We start off with a quadratic estimating valley and end up with a hill whose top is close to the local maximum at $-1$ (top right). Notice the bias that exists in the hill top estimated. Control input is plotted against number of steps (bottom left). Estimated parameters $\hat{a}$, $\hat{b}$, and $\hat{c}$ are plotted to show the convergence process during estimation (bottom right).

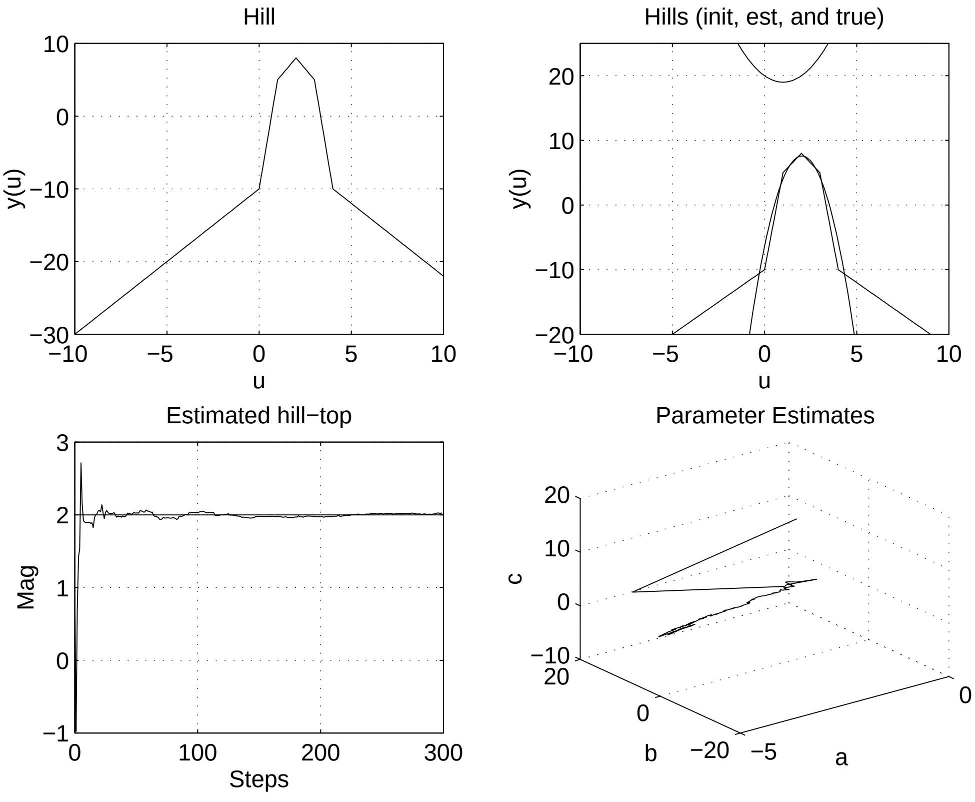


Figure 6: *Model M1 with a symmetric piecewise function as a hill*

Figure 6: This is the case where the true hill is a symmetric piecewise function of control input. Here we can see a plot of the true hill (top left). We start off with a quadratic estimating valley and end up with a hill whose top represents exactly the position of the true hill top (top right). Thus for a symmetric hill the algorithm does very well. Control input is plotted against number of steps (bottom left). Estimated parameters $\hat{a}$, $\hat{b}$, and $\hat{c}$ are plotted to show the convergence process during estimation (bottom right).

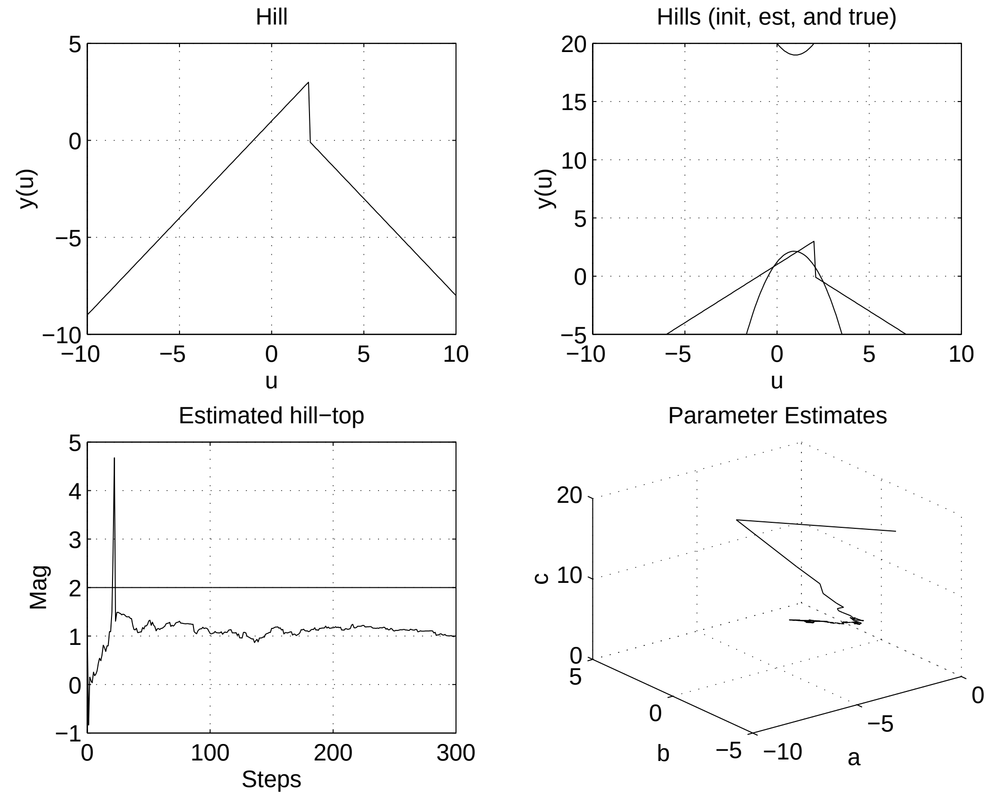


Figure 7: *Model M1 with an asymmetric piecewise function as a hill.*

Figure 7: This is the case where the true hill is an asymmetric piecewise function of control input. Here we can see a plot of the true hill (top left). We start off with a quadratic estimating valley and end up with a hill whose top has considerable bias compared with the true one (top right). Thus for an asymmetric hill the algorithm does produce biased estimate. Control input is plotted against number of steps (bottom left). Estimated parameters $\hat{a}$, $\hat{b}$, and $\hat{c}$ are plotted to show the convergence process during estimation (bottom right).

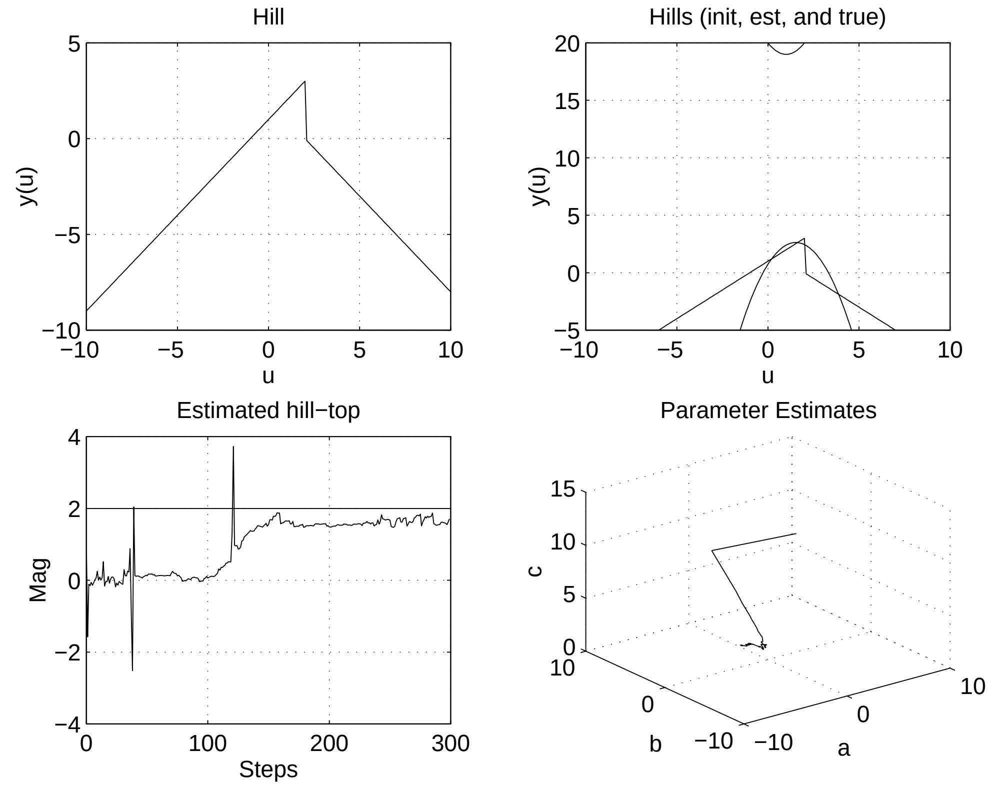


Figure 8: *Model M1 with an asymmetric piecewise function and small dither signal variance.*

Figure 8: We can see a plot of the true hill (top left). We start off with a quadratic estimating valley and end up with a hill whose top has considerable bias compared with the true one (top right). Control input is plotted against number of steps (bottom left). Notice that the bias is now reduced. Thus smaller dither signal produce less bias. Estimated parameters $\hat{a}$, $\hat{b}$, and $\hat{c}$ are plotted to show the convergence process during estimation (bottom right).

Here (Figure 8) the hill is the same one as Figure 7. Smaller dither signal is used ($\sigma_v^2 = 0.05$ compared to $\sigma_v^2 = 0.5$ in Figure 7).

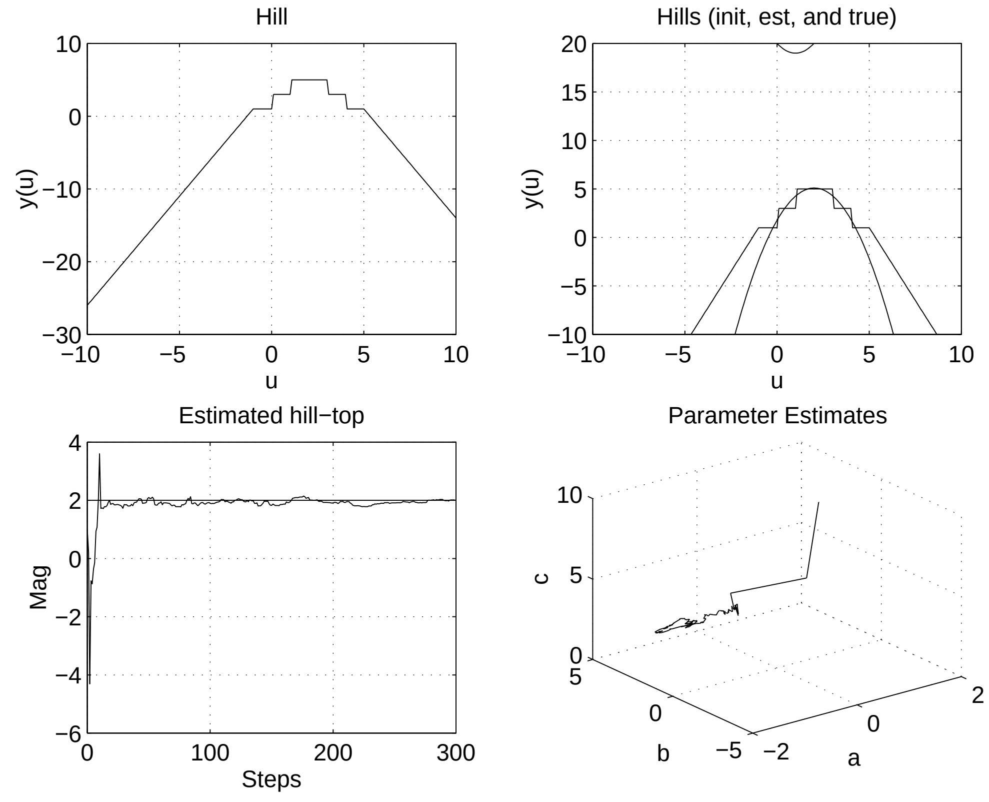


Figure 9: *Model M1 with a hill that has flat top.*

Figure 9: The hill now has got flat parts and its top is also flat. The hill itself is symmetric. The result has got some bias but converge to the position of the hill top. We can see a plot of the true hill (top left). We start off with a quadratic estimating valley and end up with a hill (top right). Control input is plotted against number of steps (bottom left).

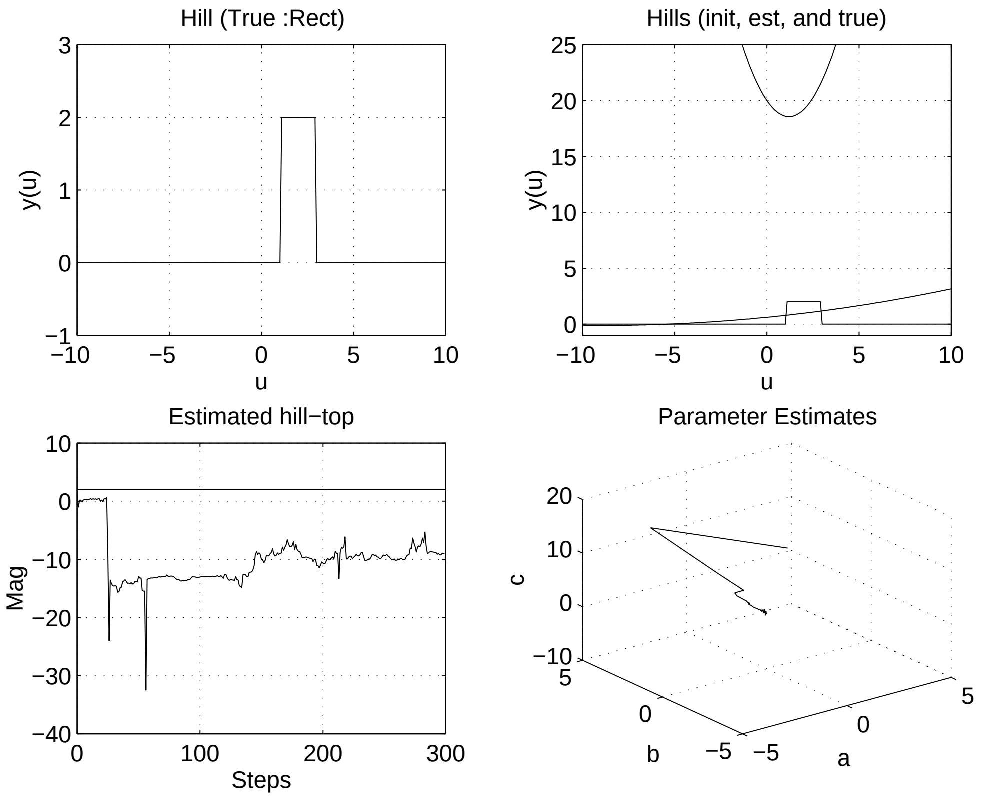


Figure 10: *Model M1 with pulse-shaped hill*

Figure 10: We can see a plot of the true hill (top left). We start off with a quadratic estimating valley and end up with a hill (top right). Control input is plotted against number of steps (bottom left). Estimated parameters $\hat{a}$, $\hat{b}$, and $\hat{c}$ are plotted to show the convergence process during estimation (bottom right).

The hill in Figure 10 gives no directional information outside its range. The hill itself is symmetric. Sometime the result does converge but here we have shown the case where it does not converge to the true position after 300 time steps.

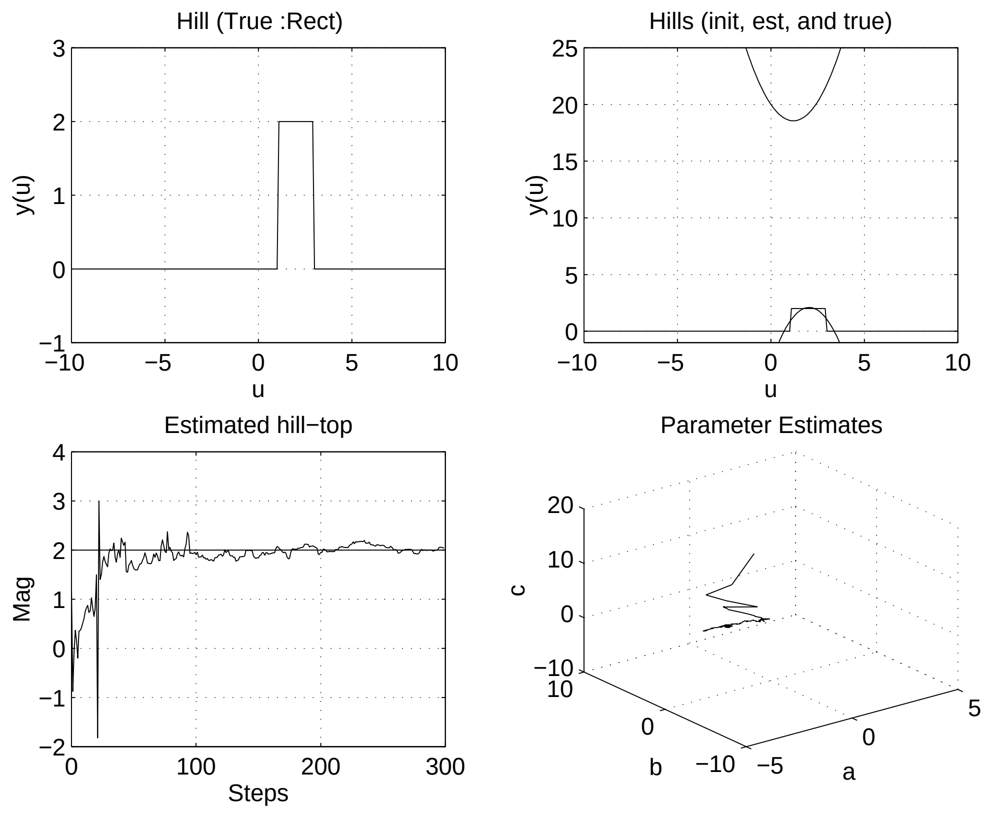


Figure 11: *Model M1 with pulse-shaped hill (another one)*

Figure 11: This is the case where we can get the correct estimation.

### 2.1.5 Discussion

We have investigated the influence of dither signal variance on estimation of the hill top. From (Figure 2) we can see that if the dither signal is small it will take longer before the estimates can reach the final value. Hence the phase plane shows a larger hazy patch around the true value of states.

The quality of estimation depends on many factors. If the true hill is badly conditioned, as an example a non-symmetric hill, there may be some bias or some difficulties in convergence. Also if the hill rises from a flat plane, the estimating procedure might get lost when not on the hill. Here is where a random search or some other procedure must be introduced to find the hill again.

### 2.1.6 Summary

A quadratic model is a suitable model for estimating a hill top because it is a good representation of the summit in the close neighbourhood of extremum. If we choose our dither signal carefully we will be able to get the accuracy required provided that the environment is not too noisy and the hill is reasonably well conditioned.

## 2.2 Reduced-parameter model

### 2.2.1 Introduction

The number of estimation parameters can be reduced from three to one. Note that (4) does not require $\hat{c}(t)$ and this parameter can be removed from (1) by differencing the data as follows. Let

$$\Delta y(t) = y(t) - y(t-1) \tag{9}$$

then

$$M2: \quad \Delta y(t) = a\Delta u^2(t-1) + b\Delta u(t-1) \tag{10}$$

Secondly we may fix $a = a_f < 0$ and (10) may be written as

$$y'(t) = \mathbf{x}^T\theta \tag{11}$$

where

$$\begin{aligned} y'(t) &= \Delta y(t) - a_f\Delta u^2(t-1) && (12)\\ \mathbf{x}(t) &= \Delta u(t-1) && (13)\\ \theta &= b && (14) \end{aligned}$$

Then, based on the model

$$M3: \quad \Delta y(t) = a_f\Delta u^2(t-1) + b\Delta u(t-1) \tag{15}$$

The basic recursive algorithm is as before with the modifications

. Initially choose $\hat{b}(0)$ and scalar $P(0)$

. (4) is replaced by (16)

. RLS is based on the model (11) - (14)

**Note** that, because $a_f$ may not be the true value of the curvature parameter, the success of the algorithm depends on $\hat{b}(t)$ converging to a value $b^*$ such that $-\frac{b^*}{2a_f}$ is still the correct input to achieve the summit.

### 2.2.2 Reduced-parameter model algorithm

**Initial Conditions** Set $t = 1; a_f = -7, \hat{b}(0), \hat{b}(1), P(1)$

**Step 1** Calculate

$$u(t) = -\frac{\hat{b}(t)}{2a_f} + v(t) \tag{16}$$

**Step 2**

$$t = t + 1 \tag{17}$$

$$y'(t) = bx(t) + e(t) \tag{18}$$

**Step 3**

$$P(t) = \frac{1}{\lambda}P(t-1)\left\{1 - \frac{x^2(t)P(t-1)}{\lambda + x^2(t)P(t-1)}\right\} \tag{19}$$

$$\hat{b}(t) = \hat{b}(t-1) + P(t)x(t)\left\{y'(t) - \hat{b}(t-1)x(t)\right\} \tag{20}$$

**Step 4** Goto Step 1

**NB**

$$x(t) = \Delta u(t-1) = u(t-1) - u(t-2) \tag{21}$$

#### 2.2.3 Reduced-parameter model simulation

Investigate the nature of reduced-parameter algorithm in estimating hill-top position. By discarding $c$ and holding $a$ constant $(= a_f)$ only the $b$ parameter is estimated [1].

Basis : $\lambda = .98, \sigma_e^2 = 1.0, \sigma_v^2 = 0.5, a_f = -7$

#### 2.2.4 Results

The result does not depend on initial position of estimated hill top.

**When** $\lambda = 0.10, 0.50, 0.70, 0.90, 0.95, 0.98, 0.99, 1.0, 3.0$ **were used.** $\lambda$ between 0.95 and 1 gives good result. Smaller $\lambda$ produces much uncertainty.

**When** $\sigma_v^2 = 0.1, 0.3, 0.5, 0.7, 0.9, 1, 3, 5, 7, 9$ **were used.** Larger variance gives better estimation, ie. less uncertainty and faster estimation time.

**When** $\sigma_e^2 = 0.1, 0.3, 0.5, 0.7, 0.9, 3, 5, 7, 9$ **were used.** When $\sigma_e^2$ increases, uncertainty in estimation increases.

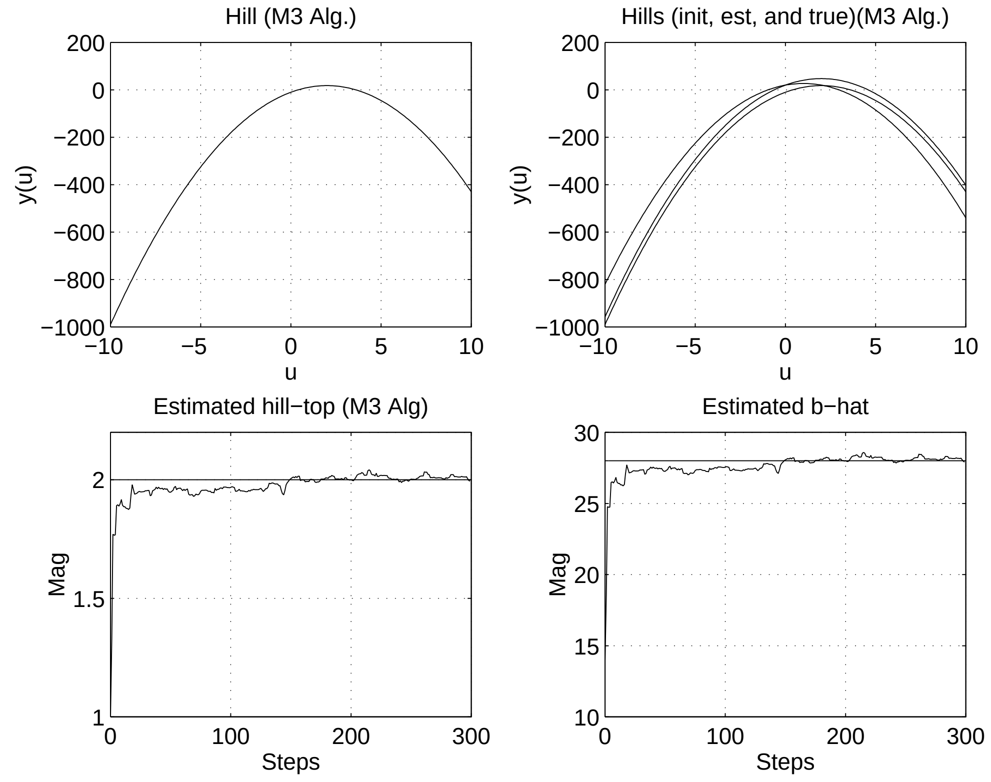


Figure 12: *Model M3 with a quadratic hill*

Figure 12: The hill used here is quadratic. We estimate only one parameter (ie. $b$). We can see a plot of the true hill (top left). We start estimating our hill arbitrarily and end up with a correct hill top position (top right). Notice that the estimated hill and the true hill are not exactly the same but their hill top positions are the same. Control input is plotted against number of steps (bottom left). Estimated parameters $\hat{a}$, $\hat{b}$, and $\hat{c}$ are plotted to show the movement during estimation procedure (bottom right).

### 2.2.5 Discussion

The reduced-parameter quadratic model seems to work better than the model with all three parameters being estimated. This must be because we assume that our $a$ parameter is a fixed value, ie. we assume that we have more knowledge about the system.

The effects of various variables, for example $\sigma_v^2$ and $\sigma_e^2$ on the estimation is the same as those of simulation using the model M1 described in the last chapter. At this stage we have not done further study in using the model to different shapes of hill. It would be an interesting thing to do.

### 2.2.6 Summary

We investigate briefly the quadratic model with reduced parameters. We have verified that it is possible to discard the $c$ parameter and to fix the $a$ parameter in our model.

## 2.3 Multi-input model

### 2.3.1 Introduction

In the case when the control input is a vector (ie. we have a number of adjustable factors that influence performance) (1) can be generalised to

$$y = \mathbf{u}^T A \mathbf{u} + \mathbf{b}^T \mathbf{u} + c \tag{22}$$

where $\mathbf{u}$ is the control vector (of dimension m) and $A$ is a symmetric matrix $(m \times m)$. If we estimate all the parameters in this model, we will have $\frac{1}{2}(m+1)(m+2)$ of them (for $m = 1$ this gives 3). We assume the model that is equivalent to (15), ie. fix $A$ and use data differencing :

$$M3' : \quad \Delta y(t) = -\sum_{i=1}^{m} \Delta u_i^2(t-1) + \sum_{i=1}^{m} b_i \Delta u_i(t-1) \tag{23}$$

This corresponds to choosing the fixed $A$ as

$$\mathbf{A}_f = -\mathbf{I}_m \tag{24}$$

(23) can then be written in the form (11), where

$$\begin{aligned} y'(t) &= \Delta y(t) + \sum_{i=1}^{m} \Delta u_i^2(t-1) &(25)\\ \mathbf{x}(t) &= [\Delta u_1(t-1), \cdots, \Delta u_m(t-1)]^T &(26)\\ \theta &= \mathbf{b} &(27) \end{aligned}$$

In this model there are only $m$ parameters to be estimated.

The control synthesis (4) (Step 1 of the algorithm) follows from (22) by calculating **u** to maximize the quadratic, ie.

$$\mathbf{u} = -\frac{1}{2}\mathbf{A}^{-1}\mathbf{b} \tag{28}$$

Introducing the assumption (24) and adding dither we get

$$\mathbf{u} = \frac{1}{2}\mathbf{b} + dither \tag{29}$$

ie.

$$u_i(t) = \frac{1}{2}\hat{b}_i(t) + v_i(t), \quad i = 1, \cdots, m \tag{30}$$

Note that there is a different dither component on each control component.

### 2.3.2 Multi-input model algorithm (Model $M3'$)

The model used is one with two inputs. Here we do not consider the $c$ parameter and do not take in to account the $a$ parameter.

**Initial Conditions** Set $t = 1; \mathbf{A}_f = -\mathbf{I}_2, \hat{b}_1(0), \hat{b}_1(1), \hat{b}_2(0), \hat{b}_2(1), \mathbf{P}_{2\times 2}(1)$

**Step 1** Calculate

$$u_1(t) = \frac{\hat{b}_1(t)}{2} + v_1(t) \tag{31}$$

$$u_2(t) = \frac{\hat{b}_2(t)}{2} + v_2(t) \tag{32}$$

**Step 2**

$$t = t + 1 \tag{33}$$

$$y(t) = g(\mathbf{x}(t)) + e(t) \tag{34}$$

**Step 3**

$$\mathbf{P}(t) = \frac{1}{\lambda}\mathbf{P}(t-1)\left\{\mathbf{I} - \frac{\mathbf{x}(t)\mathbf{x}^T(t)\mathbf{P}(t-1)}{\lambda + \mathbf{x}^T(t)P(t-1)\mathbf{x}(t)}\right\} \tag{35}$$

$$\hat{\theta}(t) = \hat{\theta}(t-1) + \mathbf{P}(t)\mathbf{x}(t)\left\{y(t) - \mathbf{x}^T(t)\hat{\theta}(t-1)\right\} \tag{36}$$

**Step 4** Goto Step 1

**NB**

$$\mathbf{x}(t) = [\Delta u_1(t-1), \Delta u_2(t-1)]^T$$

$$\hat{\theta} = \left[\hat{b}_1(t), \hat{b}_2(t)\right]^T$$

The next algorithm that we will try is the one which estimates nearly all the parameters (ie. $a$'s, $b$'s, and $c$'s). Here we only fix some of the parameters in the $A$ matrix. We are going to use this model for the object-location in the next chapter.

**Initial Conditions** Set $t = 1; \hat{a}_1(0), \hat{a}_2(0), \hat{b}_1(0), \hat{b}_2(0), \hat{c}_2(0), \hat{c}_2(0), \mathbf{P}_{6\times 6}(0)$

**Step 1** Calculate

$$p_1(t) = \frac{\hat{b}_1(t)}{2\hat{a}_1} + v_1(t) \tag{37}$$

$$q_2(t) = \frac{\hat{b}_2(t)}{2\hat{a}_2} + v_2(t) \tag{38}$$

**Step 2**

$$t = t + 1 \tag{39}$$

$$y(t) = g(\mathbf{x}(t)) + e(t) \tag{40}$$

**Step 3**

$$\mathbf{P}(t) = \frac{1}{\lambda}\mathbf{P}(t-1)\left\{\mathbf{I}_6 - \frac{\mathbf{x}(t)\mathbf{x}^T(t)\mathbf{P}(t-1)}{\lambda + \mathbf{x}^T(t)P(t-1)\mathbf{x}(t)}\right\} \tag{41}$$

$$\hat{\theta}(t) = \hat{\theta}(t-1) + \mathbf{P}(t)\mathbf{x}(t)\left\{y(t) - \mathbf{x}^T(t)\hat{\theta}(t-1)\right\} \tag{42}$$

**Step 4** Goto Step 1

**NB**

$$\mathbf{x}(t) = \left[p^2(t-1), q^2(t-1), p(t-1), q(t-1), 1, 1\right]^T$$

$$\hat{\theta}(t) = \left[\hat{a}_1(t), \hat{a}_2(t), \hat{b}_1(t), \hat{b}_2(t), \hat{c}_1(t), \hat{c}_2(t)\right]^T$$

### 2.3.3 Multi-input model simulation

Use a quadratic hill with two inputs. Study the effects of values of $\lambda$, $\sigma_v^2$, $\sigma_e^2$ and initial estimated hill top.

Basis : $\sigma_e^2 = 1, \sigma_{v1}^2 = 1, \sigma_{v2}^2 = 0.5, \lambda = 0.98$, $A$ is a matrix which all element is one

### 2.3.4 Results

**When** $\lambda = 0.70, 0.90, 0.95, 0.98, 1.0$ **were used.** $\lambda = 0.98$ and 1 give good result. With smaller value of lambda uncertainty in estimated hill top increases.

**When $\sigma_v^2 = 0.1, 3, 7$ were used.** Greater values of dither variance give more accurate results.

**When $\sigma_e^2 = 0.1, 3, 5, 10$ were used.** Smaller values of $\sigma_e^2$ give more accurate results.

**When various positions of the hill are used.** Accuracy of convergence does not depend on position of the true hill.

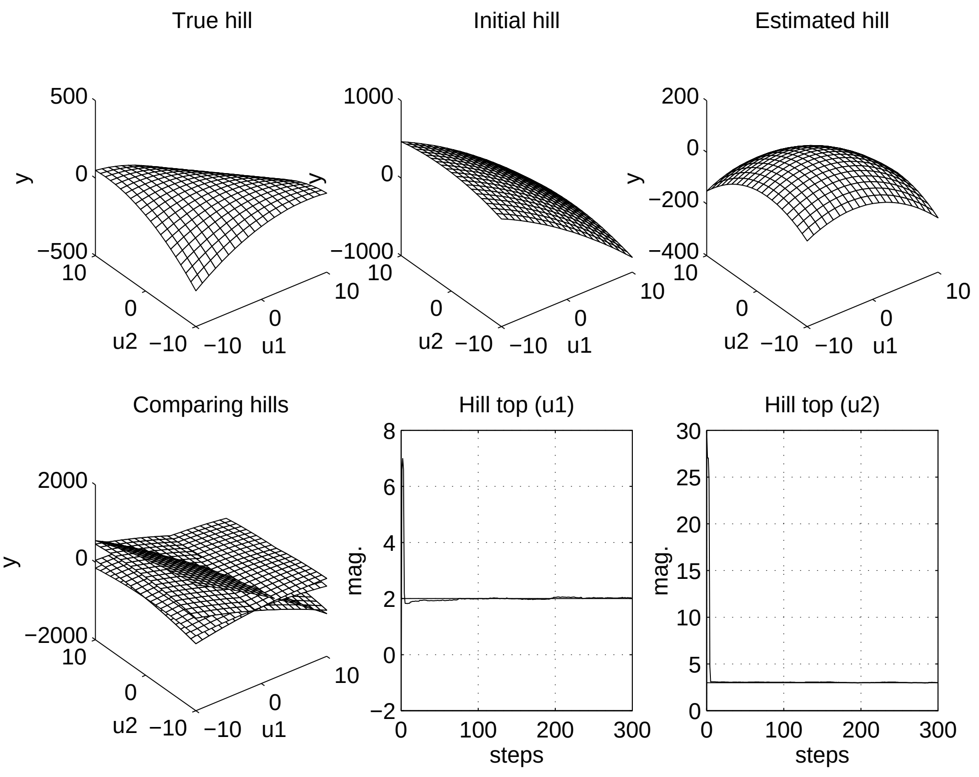


Figure 13: *Model $M3'$ with two inputs*

Figure 13: The hill used here is quadratic. We estimate only one parameter for each input (ie. $b$'s). We can see a plot of the true hill (top left), the initial hill used for estimating procedure (middle top), and the final estimated hill (top right). We start estimating our hill arbitarily and end up with a correct hill top position. The three hills are then super-imposed (bottom left). The last two plots are the estimated values of hill tops for both the control inputs (bottom middle and right)

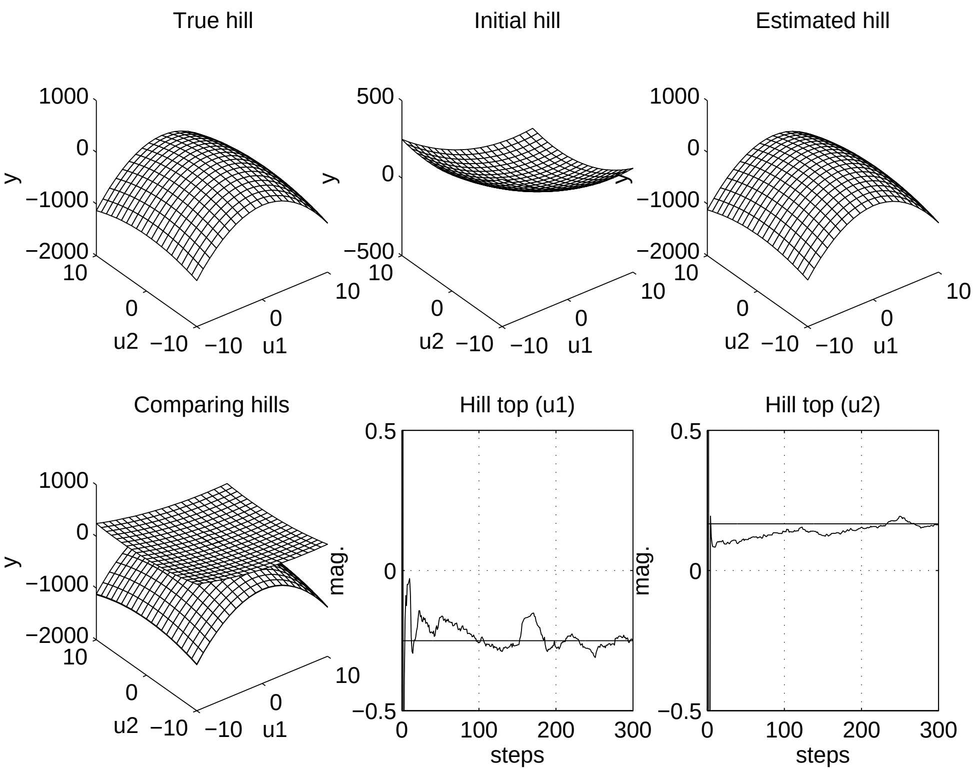


Figure 14: *Model $M3'$ with two inputs*

Figure 14: The hill used here is quadratic. We estimate all three parameters (ie. $a$, $b$, and $c$). We can see a plot of the true hill (top left), the initial hill used for estimating procedure (middle top), and the final estimated hill (top right). We choose to start with a valley and end up with a hill that has got correct hill top position. The three hills are then super-imposed (bottom left). The last two plots are the estimated values of hill tops for both the control inputs (bottom middle and right). Now we are ready to tackle the object-location problem

using this model.

### 2.3.5 Discussion

Our algorithm does very well when using with a two-input system. Again, this may be because we chose to use the reduced-parameter model. It would be interesting to investigate the same problem using full-parameter model or some other models or methods.

### 2.3.6 Summary

Often a performance function depends on more than one control value. We have shown that it is possible to extend our RLS algorithm to a multi-input quadratic model (in this case, two-input) with fixed curvature matrix.

## 2.4 Object-location model

### 2.4.1 Introduction

We may be interested in locating an object within an area (see figure 15). We consider that the object is specified by the $(x, y)$ locations of a number of pixels within the image itself. We think of the location process as a camera moving across the image and until the number of pixels seen by the camera are maximized in number. We model this as a square mask moving in the $x - y$ plane (without rotation, for simplicity) until the number of pixels within the mask is at maximum.

The model which will be used in the following algorithm has been introduced earlier in the last chapter.

### 2.4.2 Object-location model algorithm

**Initial Conditions** Set $t = 0; \hat{a}_1(0), \hat{a}_2(0), \hat{b}_1(0), \hat{b}_2(0), \hat{c}_1(0), \hat{c}_2(0), \mathbf{P}_{6\times6}(0)$

**Step 1** Calculate

$$p(t) = -\frac{\hat{b}_1}{2\hat{a}_1} + v(t) \quad (43)$$

$$q(t) = -\frac{\hat{b}_2}{2\hat{a}_2} + v(t) \quad (44)$$

**Step 2**

(p,q) is mid-point of the mask

y = overlapped pixels

**Step 3**

if y > 1 then

$$\begin{aligned} t &= t+1 && (45) \\ \mathbf{P}(t) &= \frac{1}{\lambda}\mathbf{P}(t-1)\left\{\mathbf{I} - \frac{\mathbf{x}(t)\mathbf{x}^T(t)\mathbf{P}(t-1)}{\lambda + \mathbf{x}^T(t)\mathbf{P}(t-1)\mathbf{x}(t)}\right\} && (46) \\ \hat{\theta}(t) &= \hat{\theta}(t-1) + \mathbf{P}(t)\mathbf{x}(t)\left\{y(t) - \mathbf{x}^T(t)\hat{\theta}(t-1)\right\} && (47) \end{aligned}$$

else random search for $p$ and $q$

$$\hat{b}_1 = -2a_1p \qquad (48)$$

$$\hat{b}_2 = -2a_2q \qquad (49)$$

**Step 4** Goto Step 1

**NB**

$$\begin{aligned} \mathbf{x}(t) &= \left[p^2(t-1), q^2(t-1), p(t-1), q(t-1), 1, 1\right]^T \\ \hat{\theta}(t) &= \left[\hat{a}_1(t), \hat{a}_2(t), \hat{b}_1(t), \hat{b}_2(t), \hat{c}_1(t), \hat{c}_2(t)\right]^T \end{aligned}$$

### 2.4.3 Object-location model simulation

Locating objects of various shapes and sizes at unknown locations in a predefined area. Try to match our image with the real object. Both the object and the image are two-dimensional. A square shaped image with nine-pixel size is used. The overlapped area measured is subject to white noise.

Basis : $\lambda = 0.98, \sigma_v^2 = 0.1, \sigma_e^2 = 0.1$, area = 431 pixels, object size = 9 pixels, image size = 9 pixels, N = 1000, square shaped object

### 2.4.4 Results

**When** $\lambda = 0.10, 0.30, 0.50, 0.70, 0.90, 0.95, 0.98, 0.99, 1.0, 2.0$ **were used.** $\lambda = 0.98$ and 0.99 give satisfactory convergence. With $\lambda = 0.99$ there was some bias present.

**When** $\sigma_v^2 = 0.01, 0.05, 0.1, 0.5, 1, 5$ **were used.** $\sigma_v^2 = 0.01, 0.05, 0.1, 0.5$ give satisfactory convergence with some bias.

**When** $\sigma_e^2 = 0.01, 0.05, 0.1, 0.5, 1, 5$ **were used.** $\sigma_e^2 = 0.01, 0.05, 0.1$ give satisfactory convergence. Sometime results do not converge after 1000 time steps. But normally convergence occurs after 200 steps.

**When various object shapes are used.** (square of various sizes, rectangulars of various sizes,letter-L shape, letter-O shape, letter-T shape, Plus-sign) When locating objects with a mask smaller in size, estimated position will drift but bounded inside the true object.

There is bias when using asymmetrically-shaped objects.

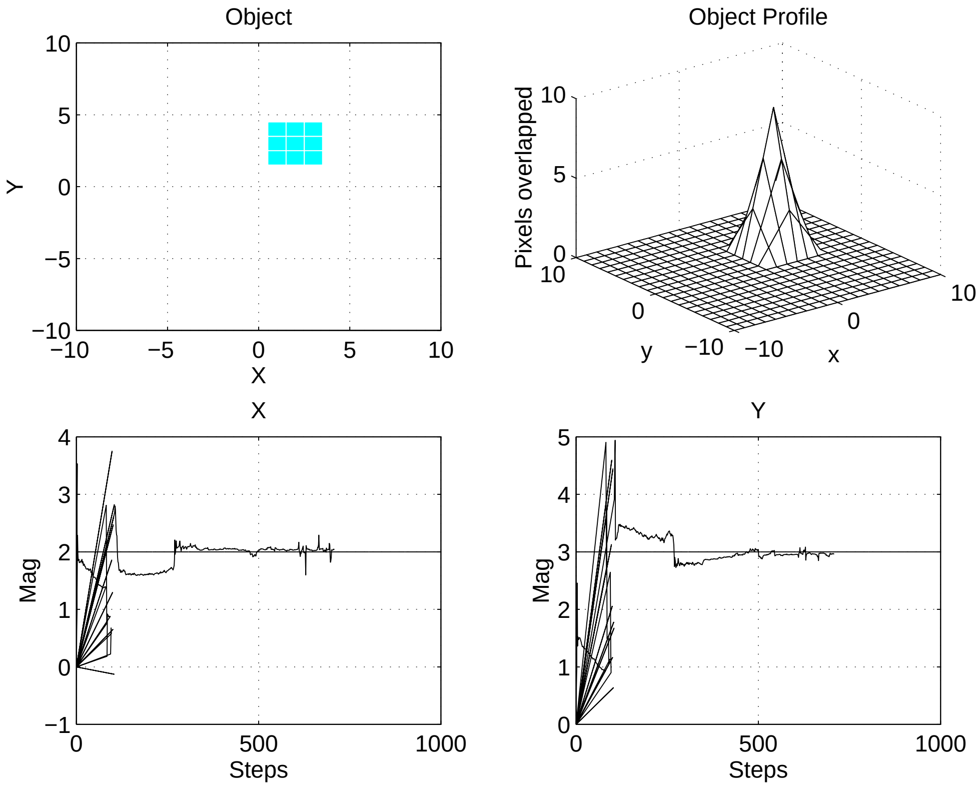


Figure 15: *Object-location when object is a square of the same size as the image.*

In Figure 15 we have a picture of the object (top left), the hill resulting from counting the number of overlapped pixels (top right), and the two coordinates of mid-point of estimated image (bottom).

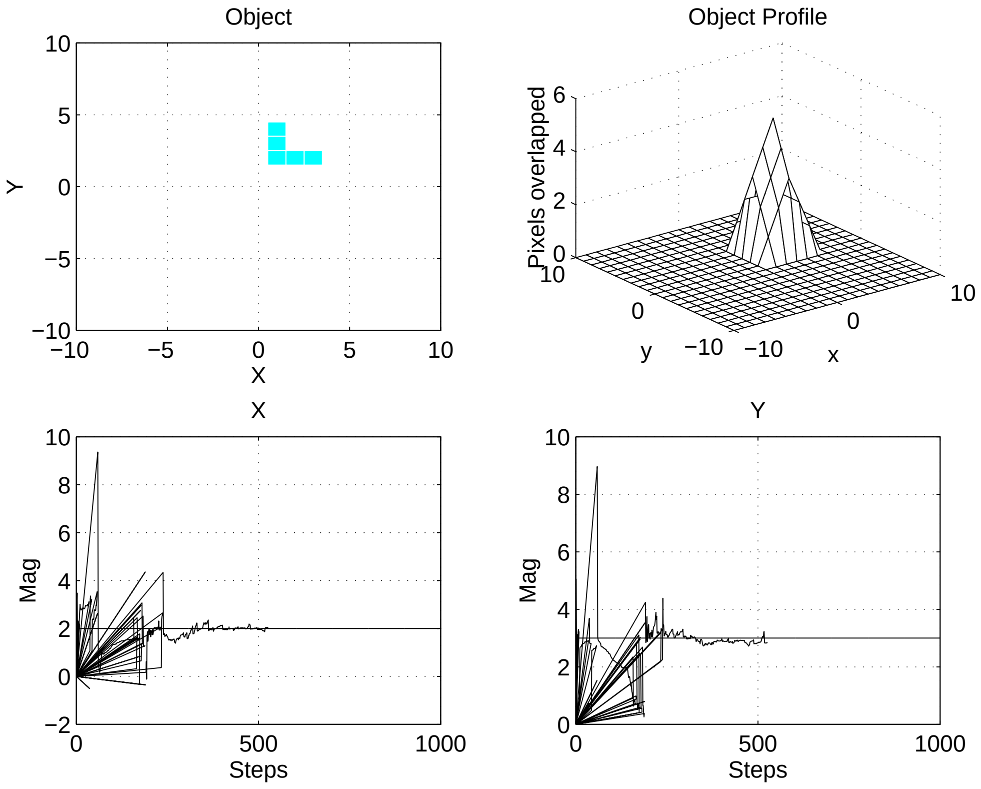


Figure 16: *Object-location when object is shaped like an L letter*

In Figure 16 we have a picture of the object (top left), the hill resulting from counting the number of overlapped pixels (top right), and the two coordinates of mid-point of estimated image (bottom).

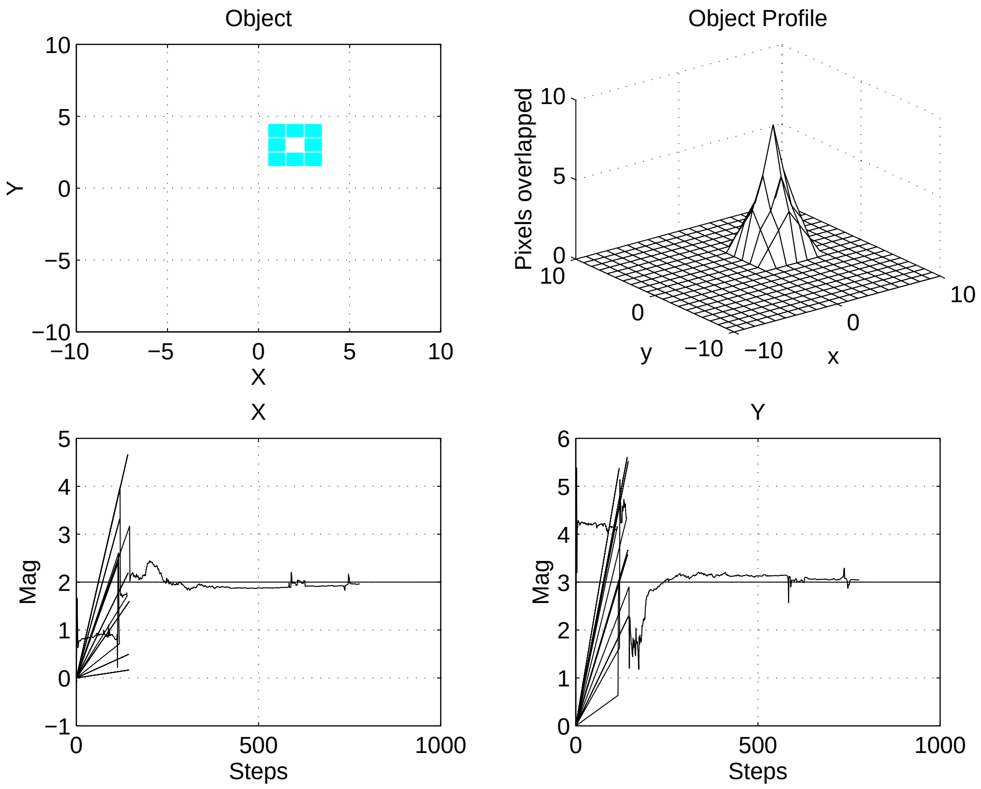


Figure 17: *Object-location when object is shaped like an O letter*

In Figure 17 we have a picture of the object (top left), the hill resulting from counting the number of overlapped pixels (top right), and the two coordinates of mid-point of estimated image (bottom).

### 2.4.5 Discussion

Because we use a random-search method in finding our way when thrown off the hill, the time of convergence does not depend only on the RLS algorithm. There are times when the result does not converge within 1000 time steps. And there is a possibility that one might be thrown off the hill after getting on it. One possible solution to this problem could be to adopt the procedure of *stepping back* when thrown off the hill.

The method of counting pixels can be time-consuming especially when our area of interest gets much larger or when the resolution increases. In real practice this problem may be overcome if we can use an analog measurement of the overlapped area and convert it to digital signal later. For example, consider measuring the intensity of light that passes through two masks.

### 2.4.6 Summary

Possibility of using the self-tuning extremum control in object-location was investigated. We have shown that the possibility is there. In order to be able to apply the idea to real problems there has to be a further and extensive study on the topic.

## 3 Discussion

For the Object Location Problem because a random search is used the time of convergence varies and the result may not converge after 1000 time steps.

Quality of convergence depends very much on how well conditioned the hill is. For a symmetric hill the result has no bias.

The purpose of dither signal is to give the essential excitation to the system. Theoretically a smaller dither signal is better. Big dither signals may cause some bias problem when a hill is asymmetric.

The value of the forgetting factor $\lambda$ must be $\leq 1$. $\lambda > 1$ means that the past data is more important than the one now, which is unrealistic. For good results $\lambda$ should be in the range $0.95 \leq \lambda \leq 1.0$.

## 4 Conclusion

*Performance optimization* [1] is one of the common objectives. Here we have used a model-based approach with a quadratic model. The result of simulation has shown that it is possible to climb a performance hill in a noisy environment.

Depending on how much knowledge of the hill we have at the start of our quest, we may be able to fix some of the parameters without any adverse effect on the usefulness of the result obtained [1].

We may extend our algorithm to systems with more than one input [2].

It is possible to apply the algorithm to the Object Location Problem. This could be useful in many areas, for example vision-robot and object tracking.

## Demonstration Program

The program used in doing the simulation has been written in Matlab code and has been adapted to be suitable for use as a demonstration program. It is called *extremum.*

The program is menu-driven. It includes almost everything done in doing this design exercise.

The author intends to write another program in some high-level language, for example C or C++. These languages are more flexible and thus should allow more ease in tailoring the program. Also, one can be free from the availability of the MATLAB program which could cause a problem sometime.

## References

[1] WELLSTEAD, P.E. and ZARROP, M.B. (1991). Self-Tuning Systems *John Wiley & Sons Ltd.*

[2] ZARROP, M.B. and ROMMENS, M.J.J.J. (1993). Convergence of a multi-input adaptive extremum controller *IEE Proceedings-D, Vol.140, No.2, March 1993*, pp.65-69

# Index